Optimized late binding: the SmallEiffel example

Olivier Zendra, Dominique Colnet

Optimized late binding: the SmallEiffel example.

Olivier ZENDRA, Dominique COLNET

{zendra,colnet}@loria.fr

LORIA
(INRIA - CNRS - Univ. of Nancy, France)
Campus Scientifique, BP 239,
54506 Vandœuvre-lès-Nancy Cedex
FRANCE

Keywords

dispatch, late binding, type inference, specialization, SmallEiffel

Issue

Object-oriented languages imply a great number of dispatched call sites, whose cost is thus a crucial point and must be reduced as much as possible in order to get the best performance. However, an important constraint weighs on the compilers and the techniques they use to generate such efficient programs: the performance of the compiler itself has to be high enough to allow its use in incremental development environments.

Many dispatch techniques have been studied by Driesen et al. [DHV95]. Dynamic dispatch techniques (e.g. Polymorphic Inline Caches) rely on run time or profile-guided information [HCU91, AH96]. They consist of various forms of caching at run time [UP87]. Static dispatch techniques precompute their data and code structure at compile time. Most are variants of the VFT method, which has been shown not to schedule very well on modern processors [DH96, DMM96]. Improving dispatch techniques requires having type information on the (sometimes whole) system. Type inference and code analysis algorithms thus appear as extremely useful, and have been subject to much work [DGC95, Age95, BS96]. These analyses also allow other optimization techniques, such as customization [CU89, CU90].

Solution Approach

A two-pronged approach has been chosen, which is implemented in our Eiffel compiler, SmallEiffel. First, it consists in analyzing the compiled system in order to solve as many polymorphic call sites as possible into monomorphic ones. It is then easy to implement the latter much more efficiently than the former. Since even the most powerful type inference algorithm cannot solve all polymorphic calls sites, a second step consists in implementing the remaining polymorphic sites as efficiently as possible.

This solution is based on a simple, fast, whole system type analysis, that may give non optimum information, but does so at a reasonable cost [CCZ97]. A very good tradeoff is thus reached between compilation speed and the optimality of the generated executable. Call sites which can be resolved as monomorphic are implemented with static, direct calls.

Because the latter schedule very well on modern processors and are likely to continue doing so in future ones, we also implement the remaining polymorphic ones as static, direct calls to specialized routines. The various possible routines — the number of which has been reduced by the type analysis — are selected by a binary branching tree [ZCC97].

These direct calls, unlike VFT-based ones, also enable further optimizations, such as inlining, which are crucial for speed. Although fairly simple, this system gives very good performance in terms of speed, and the increase in code size is very reasonable.

Of course, issues remain in terms of separate compilation and distribution of libraries without their source code.

References

[Age95] Ole Agesen. The Cartesian Product Algorithm: Simple and Precise Type Inference of Parametric Polymorphism. In *ECOOP'95*, volume 952 of *LNCS*, pages 2–26. Springer-Verlag, 1995.

[AH96] Gerald Aigner and Urs Hölzle. Eliminating Virtual Function Calls in C++ Programs. In *ECOOP'96*, volume 1098 of *LNCS*, pages 142–166. Springer-Verlag, 1996.

[BS96] David F. Bacon and Peter F. Sweeney. Fast Static Analysis of C++ Virtual Function Calls. In *OOPSLA'96*, pages 324–341, 1996.

[CCZ97] Suzanne Collin, Dominique Colnet, and Olivier Zendra. Type Inference for Late Binding. The SmallEiffel Compiler. In *JMLC*, volume 1204 of *LNCS*, pages 67–81. Springer-Verlag, 1997.

[CU89] Craig Chambers and David Ungar. Customization: Optimizing Compiler Technology for SELF, a Dynamically-Typed Object-Oriented Language. In *PLDI'89*, volume 24 of *SIGPLAN Notices*, pages 146–160, 1989.

[CU90] Craig Chambers and David Ungar. Interactive Type Analysis and Extended Message Splitting: Optimizing Dynamically-Typed Object-Oriented Programs. In *PLDI'90*, volume 25 of *SIGPLAN Notices*, pages 150–164, 1990.

[DGC95] Jeffrey Dean, David Grove, and Craig Chambers. Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis. In *ECOOP'95*, volume 952 of *LNCS*, pages 77–101. Springer-Verlag, 1995.

[DH96] Karel Driesen and Urs Hölzle. The Direct Cost of Virtual Function Calls in C++. In *OOPSLA'96*, pages 306–323, 1996.

[DHV95] Karel Driesen, Urs Hölzle, and Jan Vitek. Message Dispatch on Pipelined Processors. In *ECOOP'95*, volume 952 of *LNCS*, pages 253–282. Springer-Verlag, 1995.

[DMM96] Amer Diwan, J. Eliot B. Moss, and Kathryn S. McKinley. Simple and Effective Analysis of Statically-Typed Object-Oriented Programs. In *OOPSLA'96*, pages 292–305, 1996.

[HCU91] Urs Hölzle, Craig Chambers, and David Ungar. Optimizing Dynamically-Typed Object-Oriented Languages With Polymorphic Inline Caches. In *ECOOP'91*, volume 512 of *LNCS*, pages 21–38. Springer-Verlag, 1991.

[UP87] David Ungar and David Patterson. What Price Smalltalk ? *IEEE Computer*, 20(1), 1987.

[ZCC97] Olivier Zendra, Dominique Colnet, and Suzanne Collin. Efficient Dynamic Dispatch without Virtual Function Tables. The SmallEiffel Compiler. In *OOPSLA'97*, volume 32, pages 125–141, October 1997.